応用プログラミング

第2回　数学的計算、配列、文字列

メディア情報コース
平居　悠（ひらい　ゆたか）

到達目標

- C言語を通じてコンピュータの根幹部分を理解する。
- 目的に応じたプログラムを自由に設計できる。

授業計画

第１回　C言語の基本文法・言語処理系
第２回　数学的計算、配列、文字列
第３回　型変換、入出力処理
第４回　デバッグ、関数定義、変数の分離
第５回　プロトタイプ宣言、分割コンパイル
第６回　配列の受け渡し、ポインタ
第７回　まとめ、期末試験

前回の目標

C言語の簡単なコードを書いて実行できるようになる。

前回学んだこと

1. C言語の利用例と歴史
2. 言語処理系
3. C言語の基本
4. 条件分岐
5. 繰り返し処理

なぜC言語を学ぶのか？

- 多くのソフトウェアやOSはC言語で書かれている
- 処理が速く、メモリ消費量が少ない
- C言語を知っていると他のプログラミング言語の習得が容易になる

インタプリタ言語とコンパイラ言語

インタプリタ言語

Ruby
A PROGRAMMER'S BEST FRIEND

```
print “Hello, world!”
```

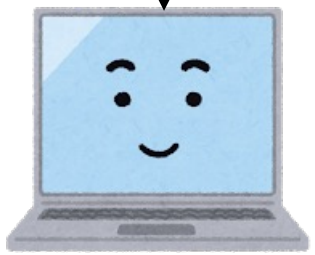
コードを1行ずつ翻訳

コンパイラ言語

C

```
#include <stdio.h>

int main()
{
   printf(“Hello, world!\n”);
   return 0;
}
```

コードを全体を翻訳



最初のプログラム

ポイント

- 大文字と小文字を区別する。
- 半角英数字で記述する（スペースも半角）。

```
#include <stdio.h>


int main()
{
    printf(“Hello, world!\n”);
    return 0;
}
```

コンパイル

```
gcc hello.c -o hello
```

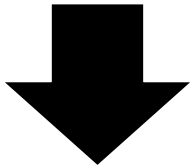
gccコマンド：コンパイルのため、言語処理系GCCを呼び出す

“-o”オプション：実行形式ファイル名を指定。
指定しないと“a.out”という実行形式ファイルが生成される。

実行

実行形式ファイルを実行

```
./hello
```



```
Hello, world!
```

と表示される。

前回の問い

- C言語はどこで利用されているのか？
  - OS、アプリ、家電製品など広く利用されている。
- C言語の基本文法はどのようになっているのか？
  - C言語はmain関数から始まる。
- C言語で条件分岐や繰り返し処理はどのように行うのか？
  - if文、for文、while文などを使う。

今回

第１回　C言語の基本文法・言語処理系
第２回　数学的計算、配列、文字列
第３回　型変換、入出力処理
第４回　デバッグ、関数定義、変数の分離
第５回　プロトタイプ宣言、分割コンパイル
第６回　配列の受け渡し、ポインタ
第７回　まとめ、期末試験

今回の問い

- C言語での演算子にはどのような規則になっているか？
- 配列とは何か？
- C言語で文字列はどのようにして扱うのか？

今回の目標

C言語で数学的な計算、複数の変数を持つデータの扱い、文字列の入出力ができるようになる。

今回学ぶこと

数学的計算
配列
文字列

今回学ぶこと

数学的計算

配列

文字列

式と演算子

演算子：特定の操作機能を有した記号 (+, −, *, /など)

オペランド：演算子の作用対象

演算子

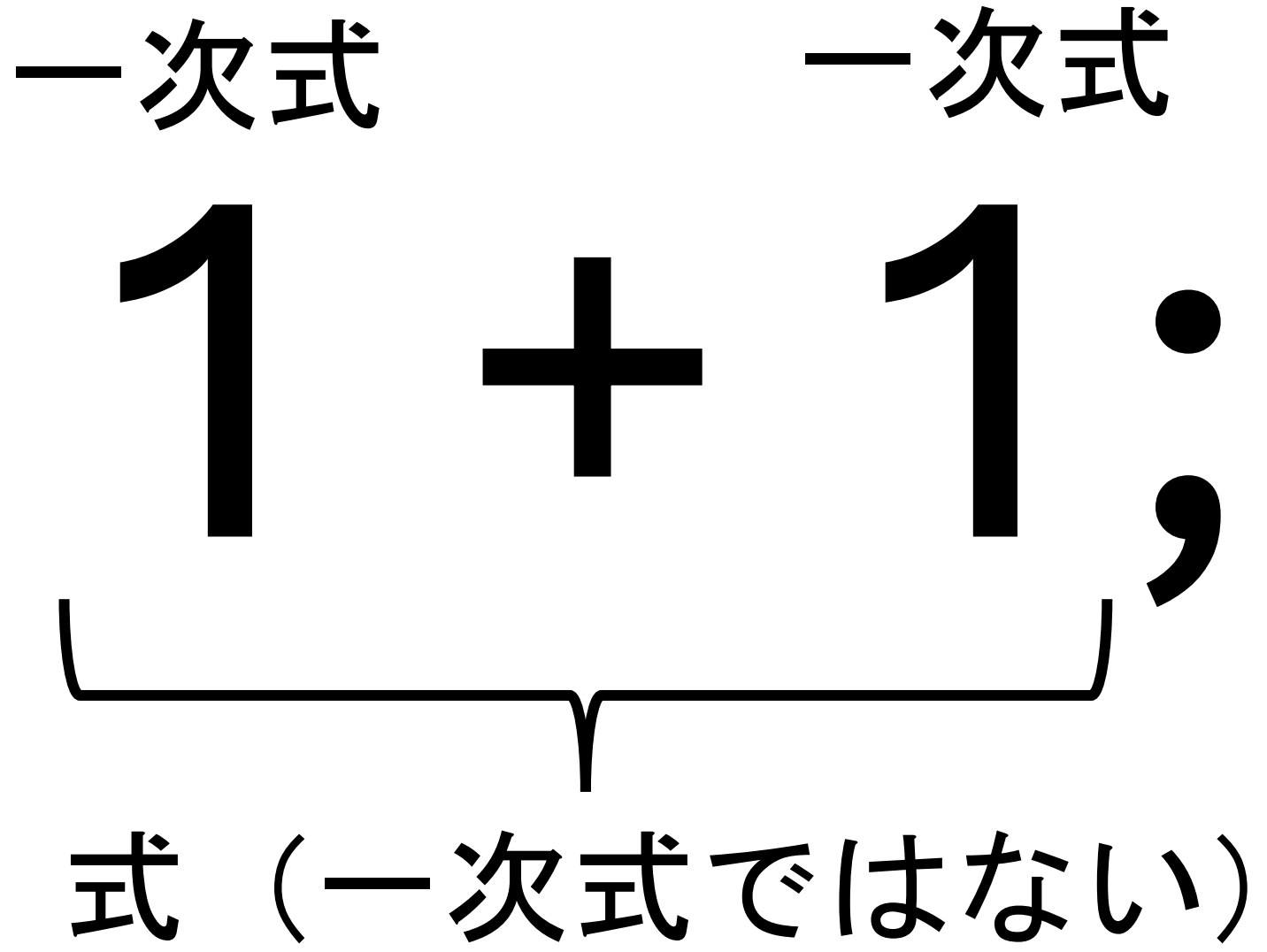
1 + 1

オペランド　　オペランド

式と演算子

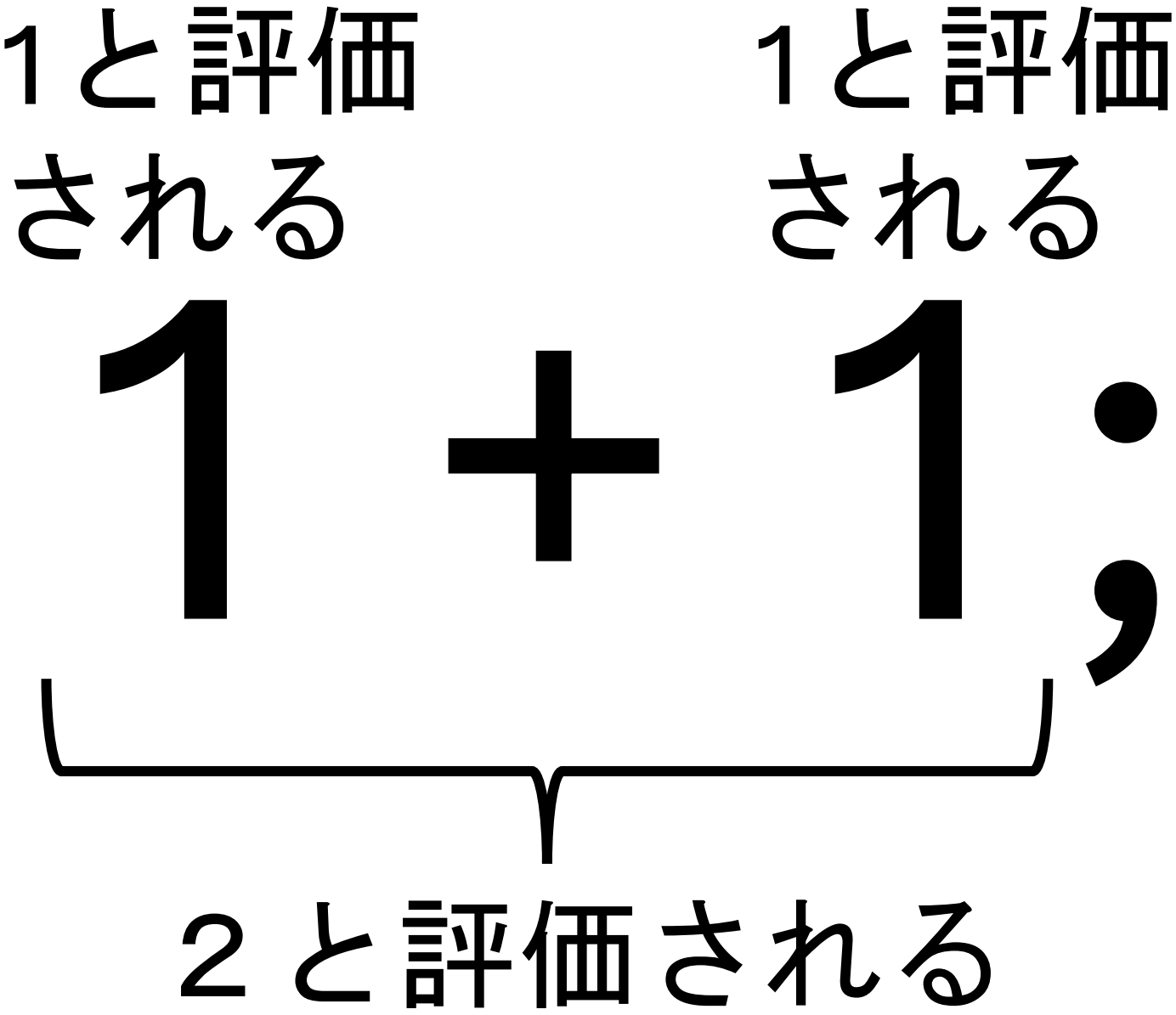
一次式：式の最も基本となる要素

式文：式のみからなる文


一次式
一次式
1 + 1;
式（一次式ではない）

式と演算子

式は**評価**されて値を持つ


1と評価される
1と評価される
1 + 1;
２と評価される

演算子の優先順位

粂井康孝『猫でもわかるC言語プログラミング』第３版、
ソフトバンククリエイティブ

優先順位	記号	意味	結合規則
1	()	関数呼び出し。printf(...), main()等	→
	[]	配列	→
	->	構造体メンバ参照	→
	.	構造体メンバ参照	→
	++	後置増分	→
	--	後置減分	→
2	++	前置増分	←
	--	前置減分	←
	sizeof	記憶量	←
	&	アドレス	←
	*	間接参照	←
	+	正符号	←
	-	負符号	←
	~	補数（ビット反転）	←
	!	否定	←
3	()	キャスト	←
4	*	乗算	→
	/	除算	→
	%	剰余	→
5	+	加算	→
	-	減算	→
6	<<	ビット左シフト	→
	>>	ビット右シフト	→
7	<	左不等号（大きい）	→
	<=	等価左不等号（以上）	→
	>	右不等号（小さい）	→
	>=	等価右不等号（以下）	→
8	==	等価	→
	!=	非等価	→
9	&	ビット積	→
10	^	ビット差	→
11	|	ビット和	→
12	&&	論理積	→
13	||	論理和	→
14	? :	条件。a ? b : cのように使う	←
15	=	代入	←
	+=	加算代入	←
	-=	減算代入	←
	*=	乗算代入	←
	/=	除算代入	←
	%=	剰余代入	←
	<<=	左シフト代入	←
	>>=	右シフト代入	←
	&=	ビット積代入	←
	^=	ビット差代入	←
	|=	ビット和代入	←
16	,	コンマ演算子	→

インクリメント・デクリメント演算子

インクリメント演算子 (++): 値を1増やす
デクリメント演算子 (--):値を1減らす

inc01.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
  int a = 10;
  int b = 10;

  printf(“a = %d\n”, a++);
  printf(“a = %d\n”, a);

  printf(“b = %d\n”, ++b);
  printf(“b = %d\n”, b);
  return 0;
}
```

aを評価した後に
aの値を1増やす

```
#include <stdio.h>

int main()
{
  int a = 10;
  int b = 10;

  printf("a = %d\n", a++);
  printf("a = %d\n", a);

  printf("b = %d\n", ++b);
  printf("b = %d\n", b);
  return 0;
}
```

bの値を1増やしてからbを評価する

```
#include <stdio.h>

int main()
{
  int a = 10;
  int b = 10;

  printf(“a = %d\n”, a++);
  printf(“a = %d\n”, a);

  printf(“b = %d\n”, ++b);
  printf(“b = %d\n”, b);
  return 0;
}
```

実行結果

a = 10

a = 11

b = 11

b = 11

sizeof演算子

式や型のサイズ（バイト）を調べる。

sizeof演算子の返す型は**size_t型**

不等号演算子

> 左辺は右辺より大きい
< 左辺は右辺より小さい
>= 左辺は右辺より大きいまたは等しい
<= 左辺は右辺より小さいまたは等しい

不等号演算子

式が真の場合、式の値は１になる
式が偽の場合、式の値は0になる

例
５＞３は真、式の値は１
５＜３は偽、式の値は０

等価演算子と非等価演算子

等価演算子 (==):
両辺が等しいときに真 (1)
非等価演算子 (!=):
両辺が等しくないときに真 (1)

代入演算子

単純代入演算子「=」

複合代入演算子:

「+=」：加算代入演算子

「−=」：減算代入演算子

「*=」：乗算代入演算子

「/=」：除算代入演算子

「%=」：剰余代入演算子

今回の問い

- C言語での演算子にはどのような規則になっているか？
  - 演算子はオペランドを伴って式を構成し、優先順位が決められている。
- 配列とは何か？
- C言語で文字列はどのようにして扱うのか？

今回学ぶこと

数学的計算

配列

文字列

配列の基本

配列：変数の集まり

宣言方法：

データ型　配列名[配列の大きさ];

例：

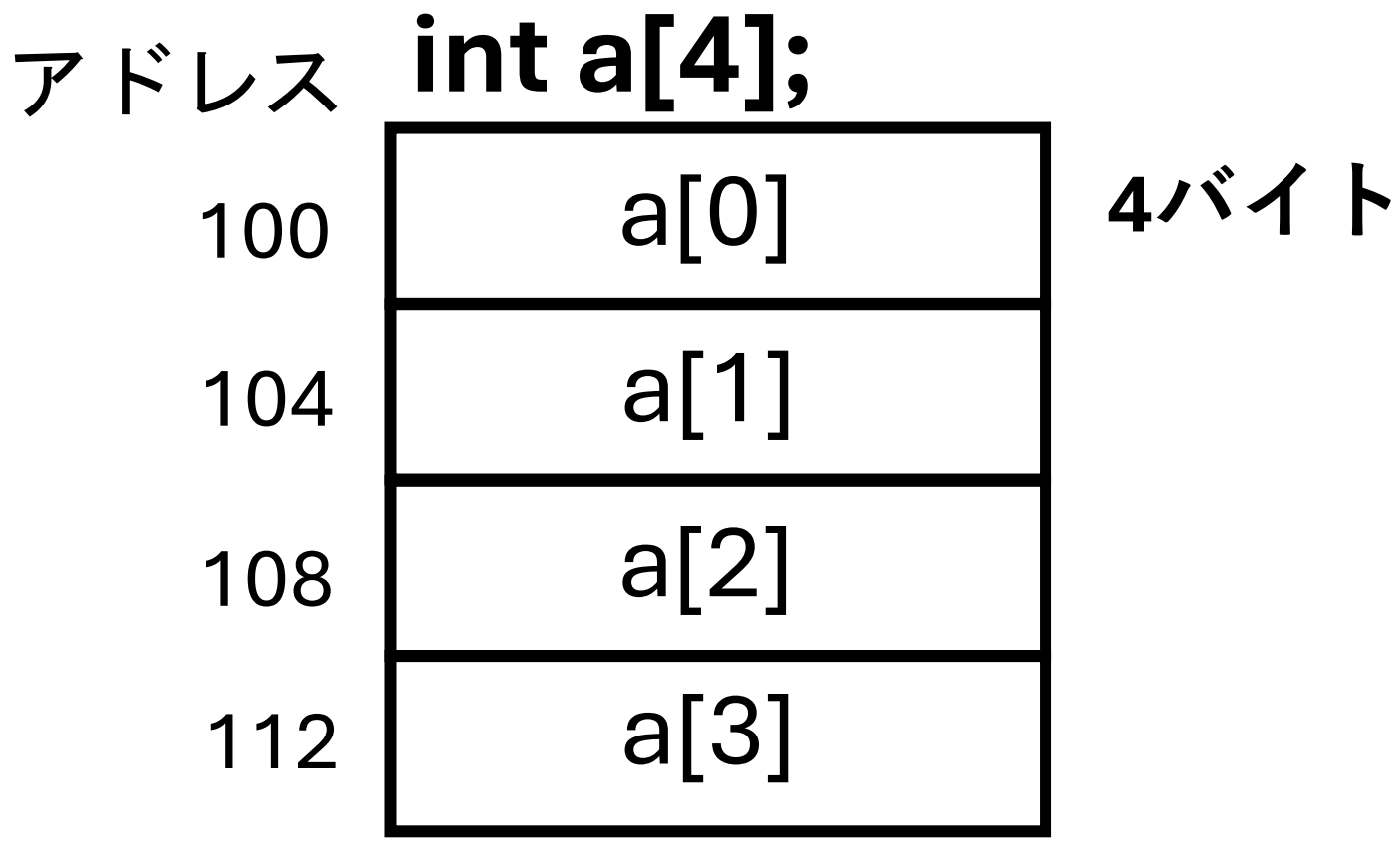
int a[4];

配列の基本

int a[4];と宣言すると、a[0], a[1], a[2], a[3]をそれぞれint型の変数として扱える。これを配列の**要素**という。[]内の0, 1, 2, 3は配列の**添え字**という。

配列とアドレス

a[0], a[1], a[2], a[3]の順でメモリ上に隙間なく並んでいる



配列のメモリ内の状態

array01.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
  int a[] = {1, 2, 3, 4}, i;

  for  (i = 0; i < 4; i++) {
    printf(“a[%d] = %d\n”, i, a[i]);
    printf(“a[%d] = %p\n”, i, &a[i]);
  }
  printf(“a = %p\n”, a);

  return 0;
}
```

配列は宣言と同時に初期化できる

宣言と同時に初期化する場合、要素数は省略できる

```
#include <stdio.h>

int main()
{
  int a[] = {1, 2, 3, 4}, i;

  for  (i = 0; i < 4; i++) {
    printf("a[%d] = %d\n", i, a[i]);
    printf("a[%d] = %p\n", i, &a[i]);
  }
  printf("a = %p\n", a);

  return 0;
}
```

配列名は、その配列の先頭要素のアドレスに変換される

```
#include <stdio.h>

int main()
{
  int a[] = {1, 2, 3, 4}, i;

  for  (i = 0; i < 4; i++) {
    printf(“a[%d] = %d\n”, i, a[i]);
    printf(“a[%d] = %p\n”, i, &a[i]);
  }
  printf(“a = %p\n”, a);

  return 0;
}
```

多次元配列

次元が２以上の配列

例：

```
int a[5][2];
int a[5][2][6];
```

多次元配列の初期化方法

```
int a[5][2] = {{0, 1}, {2, 3}, {4, 5}, {6, 7}, {8, 9}};
```

```
int a[][2] = {{0, 1}, {2, 3}, {4, 5}, {6, 7}, {8, 9}};
```

ともかける。最初の要素数のみ省略可。

多次元配列のメモリに並ぶ順番

int a[3][2];

a[0][0]

a[0][1]

a[1][0]

a[1][1]

a[2][0]

a[2][1]

一番外側（右側）から順にメモリに並ぶ

多次元配列の初期化方法

```
int a[5][2] = {{0, 1}, {2, 3}, {4, 5}, {6, 7}, {8, 9}};
```

```
int a[][2] = {{0, 1}, {2, 3}, {4, 5}, {6, 7}, {8, 9}};
```

ともかける。最初の要素数のみ省略可。

array02.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
  int a[3][2] = {{10, 20}, {30, 40}, {50, 60}};
  int i, j;

  /* 各要素の値を確かめる */
  for  (i = 0; i < 3; i++) {
    for(j = 0; j < 2; j++) {
      printf("a[%d][%d] = %d\n", i, j, a[i][j]);
    }
  }

  printf("\n");

 /* 各要素のアドレスを確かめる */
  for  (i = 0; i < 3; i++) {
    for(j = 0; j < 2; j++) {
      printf("&a[%d][%d] = %p\n", i, j, &a[i][j]);
    }
  }

  return 0;
}
```

今回の問い

- C言語での演算子にはどのような規則になっているか？
  - 演算子はオペランドを伴って式を構成し、優先順位が決められている。
- **配列とは何か？**
  - **配列は変数の集まりである。**
- C言語で文字列はどのようにして扱うのか？

今回学ぶこと

1. 数学的計算
2. 配列
3. **文字列**

文字

- 文字は‘A’, ‘B’のようにシングルクォーテーションで囲んで表現する
- データ型は**char**
- printf関数で文字出力するときの変換指定子は**%c**

char01.cというファイルを作成し、右のコードを書いて実行してみよう

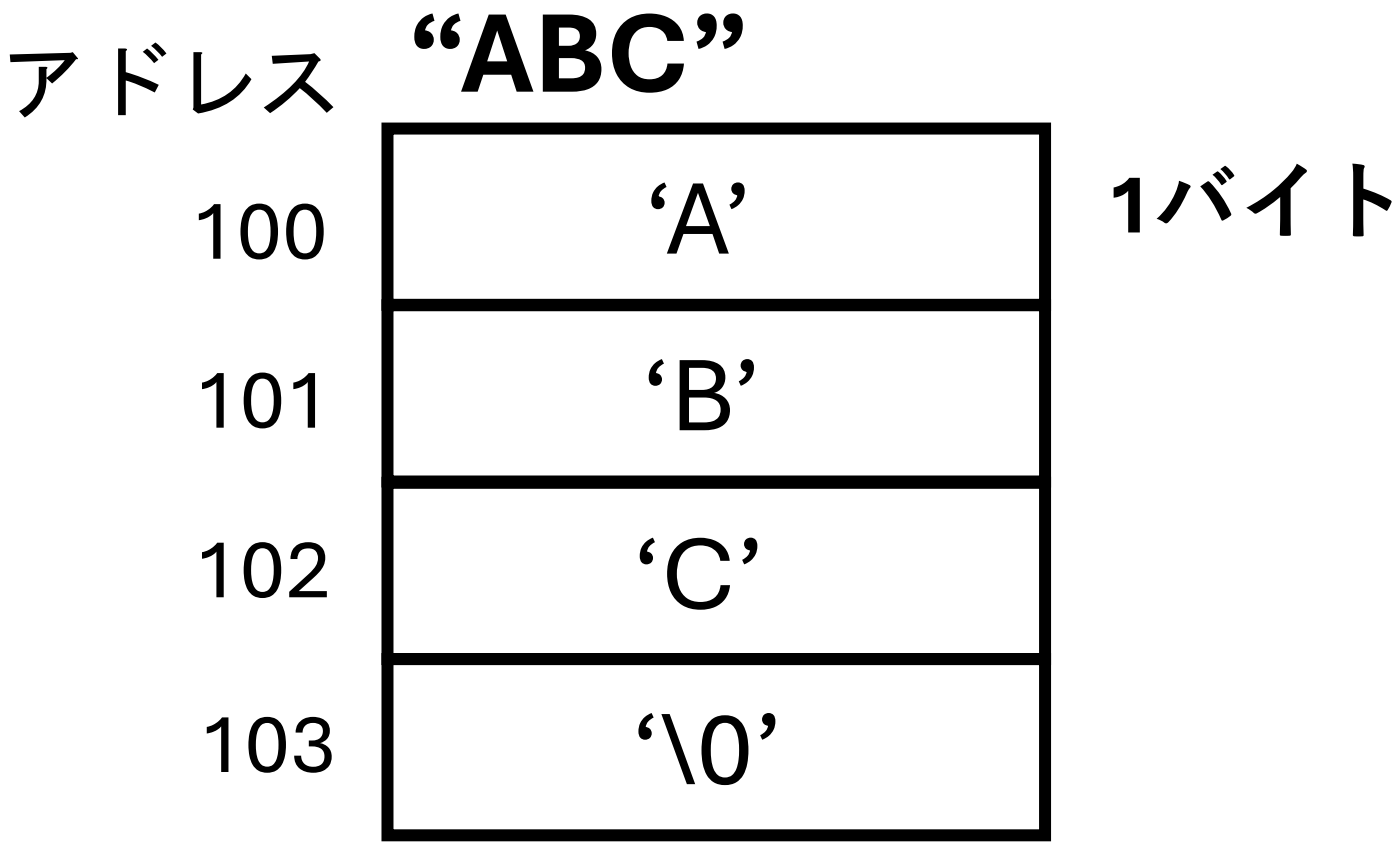
```
#include <stdio.h>

int main()
{
  char a1, a2, a3;

  a1 = ‘a’;
  a2 = ‘b’;
  a3 = ‘c’;

  /* 文字出力 */
  printf(“%c%c%c\n”, a1, a2, a3);

  /* ASCIIコード出力 */
  printf(“a1 = %d, a2 = %d, a3 = %d\n”, a1, a2, a3);

  return 0;
}
```

文字列

- ABCのような**文字列**という。
- “ABC”とダブルクォーテーションで囲ったものを**文字列リテラル**という。
- printf関数で文字列出力するときの変換指定子は**%s**

文字列とアドレス

- 文字列の式の値は先頭文字の置かれたアドレス
- 文字配列最後の要素は必ず **‘\0’** (**ヌル文字**)



配列のメモリ内の状態

文字配列

文字列をchar型の配列で扱うことができる。

文字列の入力

ユーザーに文字列を入力してもらう場合、以下例のようにする。

```
char str[16];
scanf(“%s”, str);
```

(注) strに&は不要。strは配列の名前で、先頭要素のアドレスに変換されるため。

fgets関数

入力から文字列を読み込む

fgets(文字列を受け取る配列のアドレス, 最大読み込み文字数, ファイルポインタ)

- ファイルポインタには標準入力（キーボード）から読み取る場合は**stdin**を指定する。
- fget関数はエンターキーの入力 (\n)も読み取ってしまうため、これが不要な場合は次ページのプログラムのように **‘\0’**に置き換える。

fgets関数のメリット

最大読み込み文字数を予め指定しているので、ユーザーが入力を受け取る配列のサイズより大きい文字列を入力しても指定された文字数しか受け取らないので、エラー（**オーバーフロー**）が起きない。

strinput02.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>

int main()
{
  char str[32];

  printf(“文字列を入力してください---”);
  fgets(str, sizeof(str), stdin);
  /* str配列末尾文字を ‘\0’に置き換える。*/
  str[strlen(str) - 1] = ‘\0’;

  printf(“入力した文字列は「%s」ですね。\n”, str);

  return 0;
}
```

strcpy関数

文字列をコピーする

strarray02.cというファイルを作成し、右のコードを書いて実行してみよう

```
#include <stdio.h>
#include <string.h>

int main()
{
  char str[] = “ABC”;

  printf(“%s\n”, str);
  strcpy(str, “def”);
  printf(“%s\n”, str);

  return 0;
}
```

今回の問い

- C言語での演算子にはどのような規則になっているか？
  - 演算子はオペランドを伴って式を構成し、優先順位が決められている。
- 配列とは何か？
  - 配列は変数の集まりである。
- **C言語で文字列はどのようにして扱うのか？**
  - **文字列も配列として扱うことができる。**

今回学んだこと

数学的計算
配列
文字列

今回の目標

C言語で数学的な計算、複数の変数を持つデータの扱い、文字列の入出力ができるようになる。

課題

以下の表はTaroさん、Hanakoさん、Takashiさんの英語と数学のテストの点数である。配列を用いて３名の名前と英語・数学の点数、平均点をそれぞれ表示するプログラムを作成せよ。生成AIでのコード生成は不可。

名前	英語	数学
Taro	90	60
Hanako	70	80
Takashi	50	40

提出先：yutaka.hirai@koeki-u.ac.jp

件名：応用プログラミング第2回課題 [学籍番号]

締め切り：10月15日 (木)

次回

第１回　C言語の基本文法・言語処理系
第２回　数学的計算、配列、文字列
第３回　型変換、入出力処理
第４回　デバッグ、関数定義、変数の分離
第５回　プロトタイプ宣言、分割コンパイル
第６回　配列の受け渡し、ポインタ
第７回　まとめ、期末試験